Lehrstuhl Angewandte Informatik IV
Datenbanken und Informationssysteme
Prof. Dr.-Ing. Stefan Jablonski

Institut für Angewandte Informatik
Fakultät für Mathematik, Physik und Informatik
Universität Bayreuth

Seminar

Vorname Nachname

August 26, 2015
Version: Draft / Final

Universität Bayreuth

Fakultät Mathematik, Physik, Informatik
Institut für Informatik
Lehrstuhl für Angewandte Informatik IV

Titel / Topic

Seminar

Vorname Nachname

1. Reviewer Prof. Dr.-Ing. Stefan Jablonski
Fakultät Mathematik, Physik, Informatik
Universität Bayreuth

2. Reviewer Dr. Stefan Schönig
Fakultät Mathematik, Physik, Informatik
Universität Bayreuth

Supervisors Stefan Schönig and Lars Ackermann

August 26, 2015

Vorname Nachname
Seminar
Titel / Topic, August 26, 2015
Reviewers: Prof. Dr.-Ing. Stefan Jablonski and Dr. Stefan Schönig
Supervisors: Stefan Schönig and Lars Ackermann

Universität Bayreuth
Lehrstuhl für Angewandte Informatik IV
Institut für Informatik
Fakultät Mathematik, Physik, Informatik
Universitätsstrasse 30
95447 Bayreuth
Germany

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Abstract (different language)

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Acknowledgement

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

i really wanna stay at your house

Contents

1 Introduction

1.1 Motivation

Machine learning (ML) has become a pivotal tool in decision-making processes across numerous domains, including healthcare, finance and hiring. However, as these models increasingly influence critical decisions, questions about their fairness and the ethical implications of their use have come to the forefront. Fairness in ML concerns the equitable treatment of individuals or groups, particularly in the presence of sensitive attributes such as gender, age, race, or socioeconomic status. These attributes often correlate with historical inequalities or systemic biases embedded in the data. ML models, designed to optimize accuracy, learn patterns from this data and may inadvertently exploit these unfair patterns to make predictions. While leveraging such patterns can improve predictive performance, it also risks perpetuating or even amplifying existing inequities. A notable example of this occurred when Amazon's AI recruiting tool, which was designed to assist in hiring decisions, exhibited significant gender bias, favoring male candidates due to biases in historical hiring data, leading to its eventual scrapping. [Cha23]

These challenges of fairness and bias are not confined to traditional applications but also extend to the field of predictive business process monitoring (PBPM). Here, ML models play an integral role in making decisions, whether by predicting outcomes, allocating resources, or streamlining operations. Addressing fairness in this context requires careful consideration of bias's dual nature. While biases can lead to discrimination, not all bias is inherently harmful. In some cases, certain biases may be necessary for the model to achieve its intended purpose. For instance, sensitive attributes like gender can carry significant information relevant to specific contexts. In the domain of healthcare, gender differences in biological and hormonal factors are crucial for determining effective treatments and drug prescriptions. However, the same attribute might lead to discriminatory outcomes in unrelated contexts, such as predicting whether a patient's request for treatment will be approved. This duality highlights the complexity of fairness in ML: biases that are helpful in one scenario can become harmful in another.

A significant obstacle in addressing this issue is the inherent opacity of ML models. Many models, especially complex ones such as deep neural networks, operate as

"black boxes" that produce predictions without providing insight into how those predictions are made. This lack of interpretability makes it challenging for human stakeholders and domain experts to identify whether a model's reliance on a sensitive attribute aligns with ethical and operational goals. This presents a dilemma: either leaving potentially harmful biases unchecked to avoid significantly compromising the model's predictive performance, or removing sensitive attributes entirely from the model's input. The latter approach, while intended to prevent discrimination, can lead to suboptimal predictions, especially when the sensitive attributes carry critical information relevant to the task.

1.2 Problem Statement

To address this challenge, this thesis proposes an approach based on knowledge distillation. Knowledge distillation involves transferring the encapsulated knowledge from a complex model to a simpler, more interpretable representation. By applying this technique, the inner workings of a predictive ML model can be visualized and understood by human stakeholders and domain experts, which in turn enables the identification of inherent biases in the model. Specifically for the proposed approach, the distilled representation is modified to remove the unwanted bias by adjusting the way sensitive attributes influence the representation's decision-making process. This modified version of the representation is then used to relabel the training data, creating a refined dataset that reflects the desired fairness characteristics. Finally, the original model is fine-tuned using this new, bias-adjusted training data, allowing it to learn from the corrected representation and ultimately make more equitable predictions.

The goal of this approach is twofold: to make the ML model fairer by eliminating the biases that result in unfair treatment and to maintain its predictive accuracy by preserving helpful ones. By balancing these objectives, the proposed methodology seeks to create models that are not only effective but also aligned with given ethical standards and societal expectations.

1.3 Thesis Outline

Building on the problem statement, the second chapter provides insight into existing research concerning the field of bias reduction and fairness in PBPM, while highlighting the gaps this thesis aims to address.

The third chapter provides background information essential for understanding the research. It introduces PBPM, explains the ML models used in this thesis and discusses fairness in ML.

The fourth chapter details the methodology employed in the thesis. It describes the process of data gathering and preprocessing, with a focus on handling bias causing attributes. The chapter then explains how the initial ML model is trained, followed by the application of knowledge distillation to create an interpretable representation of the model, which can then be modified to address unwanted biases. The methodology further includes how the predictions of the modified representation can be used to finetune the model to improve fairness, while maintaining accuracy.

TODO The fifth chapter presents the evaluation of the proposed approach. This chapter defines the metrics used for evaluation and includes empirical results from multiple case studies.

TODO The final chapter concludes the thesis by summarizing the findings and discussing their implications for fairness in ML. It acknowledges the limitations of the research and offers suggestions for future work.

2 Related Work

In recent years, the field of fairness in machine learning has gained significant attention, aiming to address biases and ensure equitable outcomes in predictive systems. Numerous approaches to quantifying and achieving fairness have been proposed, targeting various steps in the ML pipeline. These methods are too plentiful to go over in this thesis, but an overview can be found in [CH24].

Comparatively, the existing literature related to fairness specifically in PBPM is quite limited, although many foundational concepts and challenges concerning fairness can be transferred from general ML, as done in [DAFST22]. Accordingly, the next section will focus on reviewing approaches that have explicitly been adapted into the field of PBPM.

2.1 Fairness in Predictive Business Process Monitoring

[QA19] represents one of the initial efforts to incorporate fairness into PBPM. This approach employs discrimination-aware decision trees [KCP10], which extend traditional decision trees by adding fairness constraints during the split-selection process. These constraints penalize splits influenced by sensitive attributes, ensuring that decisions are less biased. To further enhance fairness, a relabeling technique is employed, where outcome classes of the leaf nodes are adjusted to reduce the discriminatory impact of sensitive attributes with minimal loss in predictive accuracy. This technique is similar to our methodology as described in section 4.6, there is however a key difference: The relabeling in this method is performed automatically, based on the assumption that all bias is undesirable and should be eliminated. In contrast, our approach emphasizes human stakeholder involvement to decide which biases are unwanted, allowing for the preservation of nuanced, context-dependent correlations.

In [LP24], biases in predictive models are being adressed by leveraging adversarial learning [Goo+14], a method where two models are trained simultaneously: a predictor and an adversary. The predictor model focuses on accurate outcomes, while an adversary model tries to identify sensitive attributes based on the output of

the predictor. The predictor is penalized when the adversary succeeds, encouraging fairness by reducing reliance on these attributes. While effective at reducing bias, this penalty-based approach lacks context sensitivity, treating all biases as inherently undesirable. This indiscriminate removal of biases may unintentionally eliminate meaningful correlations, thereby diminishing the predictive model's utility in specific scenarios where such relationships are crucial.

Lastly, [PDV24] focuses on ensuring independence between predictions and sensitive group memberships using a composite loss function for training the ML model. Compared to traditional loss functions, which seek to only optimize the predictive performance of a ML model, this composite loss function incorporates integral probability metrics [PZ19], in order to find a balance between accuracy and fairness. However, similar to the other approaches presented before, the uniform elimination of biases may inadvertently remove desirable correlations, potentially affecting the utility of the predictive model in certain contexts.

2.2 Conclusion

Current approaches to fairness in PBPM have made significant strides but remain somewhat limited by their automated and uniform treatment of bias. A shift toward an approach that puts more control into the hands of stakeholders presents a promising direction for achieving more context-sensitive models, that achieve better performance while still making context-based fair decisions.

3 Background

This chapter will go over definitions and background knowledge necessary for understanding the methodology and evaluation presented in later chapters. It will adress fundamentals of PBPM, go over the basics of neural networks and decision trees, and explain how fairness is quantified in this thesis.

3.1 Predictive Business Process Monitoring

Predictive Business Process Monitoring (PBPM) is a field that focuses on analyzing and forecasting the progression of ongoing business processes based on historical data. **Business processes** represent structured sets of activities or tasks undertaken by organizations to achieve specific objectives, such as processing customer orders, managing inventory or onboarding new employees. A specific execution or instance of the overall business process is refered to as a **case**. For example, in an order fulfillment process, each order corresponds to a separate case.

3.1.1 Business Process Model and Notation

In order to provide visualization for stakeholders, business processes are often portrayed in the form of **process models**. These capture the sequence and flow in a formalized representation and are generally created using the standardized **Business Process Model and Notation**, which we will also use a simplified subset of [Whi04]:

- **Activities** are represented as rectangles with rounded corners, denoting the individual tasks or operations within the process.

- **Gateways** are depicted as diamonds, used to control divergence and convergence in the workflow. These indicate either decision points or the merging of paths. In this thesis, decisions are considered to be exclusive, so only one of the paths can be chosen.

- **Events** are shown as circles, representing the initiation and completion of the process. Specifically, the start event, depicted as a circle with a narrow border,

marks the beginning of the process and the end event, depicted as a circle with a bold border, signifies the process's conclusion.

- **Sequence Flow** is represented by lines with arrowheads and is used to show the order that activities will be performed in.

3.1.2 Event Log Data

During the execution of business processes, data is captured and stored in the form of event logs. To formally define the structure of this data, let A be the universe of all process activities, C the universe of case IDs, T the universe of possible timestamps and $S_1, ..., S_m$ the universes of the m attributes associated with the process.

In this thesis, we focus exclusively on **case attributes**, which are static and remain unchanged throughout the execution of a process instance. These attributes describe characteristics of the process instance itself, rather than dynamic properties of individual events. Case attributes can be either **categorical**, containing predefined categories or classes, such as the patients gender or nationality, or **numerical**, representing continuous or discrete numerical values, like the patient's age or income.

In an event log, executed activities are recorded as individual events. Each **event** e can be formally described as a tuple $e = (a, c, s, t)$. Here, $a \in A$ is the activity that has been executed, $c \in C$ is the case ID, linking the event to a specific process instance, s is a tuple $s = (s_1, ..., s_m)$ with $s_i \in S_i, \forall i \in \{1, ..., m\}$, containing the values of each attribute and $t \in T$ is the timestamp denoting when the event occurred.

The sequence of all events $e_1, ..., e_l$ belonging to a single case c, ordered by their timestamps, is called the **trace** of c with length l. Considering that a trace describes the life-cycle of a single case c, all events belonging to a trace have the same attribute values s. A **prefix** with length l' is a portion of such a trace with length l, consisting of the first l' events, with $l' < l$. Lastly, the **event log** is a finite collection of traces belonging to the same process.

3.1.3 Next Activity Prediction

When utilizing these event logs for PBPM, there are multiple tasks one could consider taking on, such as predicting the remaining time of a case or determining its final outcome. These predictions play a crucial role in enabling organizations to optimize operations, allocate resources effectively, and preempt potential issues. In this thesis,

we will take a closer look specifically at **next activity prediction**, where the objective is to forecast the subsequent activity in an ongoing case. Formally, given a prefix $e_1, ..., e_{l'}$ of observed events, the goal is to predict the activity label $a_{l'+1}$ of the next event $e_{l'+1} = (a_{l'+1}, c_{l'+1}, s_{l'+1}, t_{l'+1})$.

3.2 Black Box vs. White Box

Within the field of PBPM, numerous traditional machine learning and deep learning architectures have been employed to develop effective predictors for next activity prediction [RMVL21]. A fundamental distinction among these approaches lies in their categorization as either black box or white box models.

Black box models, such as neural networks, excel in capturing complex patterns and relationships within data but provide limited transparency into their decision-making processes. In contrast, **white box** models, like decision trees or linear regression, prioritize interpretability and simplicity, enabling stakeholders to understand and trust the reasoning behind predictions. However, this may come at the cost of reduced predictive performance for complex tasks. [Rud19]

The difference between black and white box models will become even clearer in the following sections 3.3 and 3.4, as we will look at feedforward neural networks as an example for a black box model and decision trees as an example for a white box model.

3.3 Neural Networks

In this thesis, **feedforward neural networks** (NN) will be used as the main predictor for the task of next activity prediction. While these are not inherently designed for sequence data, such as traces in an event log, their simplicity and ease of training still makes them an attractive choice. Additionallly, the focus of this work doesn't lie on achieving maximum predictive performance, but rather the conceptual demonstration of our approach. Moreover, NNs have been employed in related work [LP24] within PBPM, demonstrating their viability for similar tasks. Nevertheless, since our methodology described in chapter 4 is not specific to using NNs, it is entirely feasible to adapt our approach to other deep learning architectures.

3.3.1 Perceptron

In order to understand NNs, it is helpful to begin with the basic computational unit in a neural network, the perceptron. The perceptron processes an input vector $x = (x_1, ..., x_n)$ in a way that is designed after biological neurons. It combines the input x linearly with a weight vector $w = (w_1, ..., w_n)$ and adds a bias b. The resulting sum is then passed to an activation function σ, which determines the final output of the perceptron. This activation function typically introduces non-linearity into the computation, such as the ReLU or sigmoid function. [**perceptron**]

Formally, the perceptron calculates its scalar output y for the input x as:

$$y = \sigma\left(\left[\sum_{i=1}^{n} w_i \cdot x_i\right] + b\right) \tag{3.1}$$

3.3.2 Network Architecture

The scalar output of a perceptron can only be used for basic binary classification or regression problems. Even then, due to the simple construction of the perceptron, its effectiveness is highly limited, depending on the complexity of the task. [**perceptron_limited**] NNs extend the perceptron model by organizing multiple perceptrons into fully connected layers, where every perceptron in one layer is connected to every perceptron in the next. This structure allows NNs to handle more complex tasks.

An NN consists of three distinct types of layers, ordered sequentially:

- Input layer: The input layer acts as the interface between raw input data and the network. Each perceptron in this layer corresponds to one feature of the input vector $x = (x_1, ..., x_n)$. The input layer does not perform any transformations, it merely transfers the raw input values to the next layer.

- Hidden layers: Hidden layers are where the network performs the majority of its computations. Each hidden layer contains a set of perceptrons that process data received from the previous layer. These layers are responsible for learning complex representations of the input data, enabling the network to capture intricate patterns and relationships.

- Output layer: The output layer produces the final predictions of the network. For multiclass classification tasks, such as next activity prediction, the layer

includes one perceptron for each class, which in our case corresponds to the number of activities.

The number of hidden layers, the number of perceptrons per layer, and the choice of activation functions collectively define the architecture of the network. These design choices are critical in determining the model's ability to handle the task at hand while balancing computational complexity and capacity.

3.3.3 Feedforward Algorithm

With the network architecture defined, the next step is to understand how input data is processed as it passes through the various layers. This process is governed by the **feedforward algorithm**.

The algorithm begins at the input layer, where the input $x = (x_1, ..., x_n)$ is passed directly to the next layer without any transformations. From there, each perceptron in the subsequent hidden layers performs the same operation as provided in formula 3.1, using the outputs from the previous layer as inputs. Formally, the activiation $y_j^{(l)}$ of the j-th perceptron in the l-th layer is computed from the m outputs $y_1^{(l-1)}, ..., y_m^{(l-1)}$ of the previous layer, using weight vector $w^{(l)} = (w_{1,j}^{(l)}, ..., w_{m,j}^{(l)})$, bias $b_j^{(l)}$ and activation function $\sigma_j^{(l)}$ as follows:

$$y_j^{(l)} = \sigma\left(\left[\sum_{i=1}^{m} w_{i,j}^{(l)} \cdot y_i^{(l-1)}\right] + b_j^{(l)}\right) \tag{3.2}$$

In the output layer, the perceptrons are activated using the softmax function. Let $z_j^{(l')} = \left[\sum_{i=1}^{m'} w_{i,j}^{(l')} \cdot y_i^{(l'-1)}\right] + b_j^{(l')}$ be the weighted sum of inputs of the j-th perceptron in the output layer L before activation. Then, given the total number of classes k, the softmax activation $y_j^{(l')}$ for the j-th output perceptron is defined as:

$$y_j^{(l')} = \frac{\exp(z_j^{(l')})}{\sum_{i=1}^{k} \exp(z_k^{(l')})} \tag{3.3}$$

The softmax activation ensures that the outputs $y_1^{(l')}, ..., y_k^{(l')}$ form a valid probability distribution, with each value between 0 and 1 and their sum equal to 1. The predicted class corresponds to the one with the highest probability.

All in all, the strictly unidirectional flow of data in the NN ensures that no feedback loops are introduced, maintaining computational simplicity. However, the interplay of numerous weights and biases across multiple layers, combined with the non-linear transformations introduced by activation functions, makes it difficult to intuitively understand or trace how specific inputs lead to specific outputs. This lack of transparency is why we consider NNs to be black box models.

3.3.4 Backpropagation Algorithm

While the feedforward algorithm enables a neural network to compute predictions by propagating input data through its layers, this process alone does not modify the network's parameters. To improve the network's predictions and reduce errors, an optimization process is required. This is achieved through the **backpropagation algorithm**, which employs the gradient descent optimization in order to iteratively update the model's weights and biases in the direction that reduces the error between the predicted and target outputs.

The training process relies on a dataset S composed of **training samples**, where each sample $(x, y) \in S$ consists of an input vector $x = (x_1, ..., x_n)$ with n numerical features and a corresponding target y. For our task of next activity prediction, y is represented as a **one-hot encoded vector** $y = (y_1, ..., y_k)$, where $y_c = 1$ for the target class c, while $y_i = 0, i \neq c$ for all other classes. The number of classes k corresponds to the amount of activities in the task.

The discrepancy between the predicted probability distribution $\hat{y} = (\hat{y}_1, ..., \hat{y}_k)$, generated by the network, and the target distribution $y = (y_1, ..., y_k)$ is quantified using the **cross-entropy loss** L. When the target distribution y is represented as a one-hot vector, the loss is given by:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i), \tag{3.4}$$

The cross-entropy loss penalizes the network heavily when the predicted probability for the target class c is low, encouraging the network to assign high probabilities to the correct classes.

Backpropagation operates in two main steps: the forward pass and the backward pass. In the forward pass, the input x is propagated through the network to compute

the predicted probabilities $\hat{y}$. The cross-entropy loss is then calculated using the predicted probabilities and the target labels.

The backward pass begins at the output layer, where the gradients of the loss function with respect to the network's predictions are directly computed based on the error calculated during the forward pass. These gradients are then propagated back through the hidden layers using the chain rule. At each layer, the gradients are used to update the weights and biases, moving them closer to their optimal values.

The forward and backward passes are repeated iteratively over the training dataset until the loss converges to a satisfactory level or a predefined number of epochs is reached. By iteratively refining the weights and biases, backpropagation enables the network to learn from data and improve its predictions over time.

3.4 Decision Trees

Similarly to NNs, we will also use **Decision Trees** (DT) as a predictor for next activity prediction. One of their most significant strengths lies in their transparency and interpretability, especially when compared to more opaque models like NNs. And despite their lack of an inherent mechanism for handling sequential dependencies, decision trees have been effectively employed in related work [QA19] within the PBPM domain, making them a suitable choice for in this thesis.

3.4.1 Tree Structure and Key Components

In order to understand DTs, we will first go over the structure and key components of a tree in general. A tree T is a special type of directed graph, defined as $T = (V, E)$, where V is the finite set of **nodes** and $E \subset V \times V$ is the set of directed **edges**. When two nodes $u, v \in V$ are connected by an edged $(u, v) \in E$, we will call u the **parent** of v and likewise v the **child** of u. Similarly, a **path** from u to v, defined a sequence of edges $(u, w_1), (w_1, w_2), ..., (w_{n-1}, w_n), (w_n, v) \subset E$, establishes u as an **ancestor** of v and v as a **descendant** of u.

Within the tree, nodes are classified into either **internal nodes**, when they have at least one child, or **leaf nodes**, when they have no children. Additionally, a node r without parents, formally satisfying $\forall v \in V : (v, r) \notin E$, denotes the **root** of the tree. Accordingly, the **subtree** rooted at a node v refers to the tree $T_v = (V_v, E_v)$, where $V_v \subset V$ is the subset containing all descendants of v and $E_v \subset E$ the edges between V_v. Another essential property of a tree is its **depth**, which measures the longest path from the root node to any of its leaf nodes.

Finally, unlike a general directed graph, a tree must satisfy several strict structural properties: [**trees**]

- **Single Root:** There has to be a unique root node $r \in V$.
- **Connectedness:** Every node $v \in V$ must be a descendant of root node r.
- **Single Parent:** Every node $v \in V$ except the root node r must have exactly one parent $u \in V$.

Furthermore, we will work exclusively with binary trees. These have the additional restriction, that every node must have either exactly two or none children. When a node has two children, we will assume a fixed distinction into a left child an a right child.

3.4.2 Decision Process

Having presented the necessary definitions, we will now take a closer look at how DTs work internally. In this thesis, input data for DTs will have the same shape as the input for NNs. Therefore a sample $x = (x_1, ..., x_n)$ is a vector containing n numerical features. When predicting the next activity y for such a sample x, the nodes of the DT serve different roles:

The root node is the starting point for all decision processes. Internal nodes posess a feature index i and a threshold value t. These internal nodes represent decision points, which examine whether the i-th feature x_i of the sample x is greater than the threshold t. Depending on the answer, the sample x is then passed onto either the left or right child. This process continues recursively until the sample reaches a leaf node. Leaf nodes contain a label y, representing the outcomes of the decision process, in our case determining the predicted next activity.

Since the structure of a DT explicitly represents the decision-making process, it becomes immediately clear why they are widely regarded as white box models. Each path from the root to a leaf corresponds to a set of human-readable decision rules, enabling stakeholders to identify important features and assess the rationale behind each decision.

3.4.3 Training Process

Now that we understand how a DT makes predictions, we turn our focus to the training process—how a DT is built from data. Let S be the collection containing the training data and $|S|$ the amount of elements contained in S. We assume the elements of S to be tuples of the form (x, y), where $x = (x_1, ..., x_n)$ is a sample input and y the corresponding target output label. The training process aims to construct a structure that effectively partitions S in a way, such that the target labels y within the partitions are as homogeneous as possible. This is achieved by selecting the feature and threshold for each split that maximize the "purity" of the resulting subsets.

A common metric used to evaluate this purity in data sets is **Gini impurity**. The Gini impurity measures the likelihood of incorrectly classifying a randomly chosen element, if it were labeled according to the distribution of labels. Intuitively, it quantifies how "mixed" the labels are within S. Mathematically, for k different target labels, where p_i is the proportion of samples in S having the i-th target label, the Gini impurity is defined as:

$$G(S) = 1 - \sum_{i=0}^{k} p_i^2 \tag{3.5}$$

The Gini impurity $G(S)$ ranges from 0 to $1 - \frac{1}{k}$. A value of $G(S) = 0$ indicates that S is perfectly pure, with all samples having a single target label, while $G(S) = 1 - \frac{1}{k}$ represents a maximally impure dataset where samples are evenly distributed among all k target labels. Generally, a lower $G(S)$ implies a higher purity. [**gini**]

The training process begins at the root node, which is associated with the entire training dataset S. At each step, the algorithm evaluates all possible splits of the data by testing every feature x_i and various thresholds. Thresholds for splitting are derived directly from the values in the dataset associated with the node, ensuring that every threshold capable of separating samples into distinct subsets is tested. Specifically, given a feature x_i and a sorted list of its values in the associated dataset, potential thresholds are chosen as the midpoints between consecutive unique values. Splits are chosen greedily, selecting the feature and threshold that provide the largest reduction in impurity. Formally, if splitting S into S_1 and S_2 results in Gini impurities $G(S_1)$ and $G(S_2)$, the split is selected to minimize the weighted Gini impurity G_{split}:

$$G_{split}(S_1, S_2) = \frac{|S_1|}{|S|} G(S_1) + \frac{|S_2|}{|S|} G(S_2) \tag{3.6}$$

Once the best split is determined, two child nodes are created, and each is associated with one of the datasets S_1 and S_2 resulting from the split. This process is recursively

repeated for each child node until a stopping criterion is met, such as reaching a maximum tree depth, achieving a minimum number of samples per leaf, or reducing impurity to a negligible level.

At the end of this recursive process, the leaf nodes of the tree represent the terminal points of the decision-making structure. Each leaf is associated with a subset of the training data that corresponds to the sequence of splits leading to that leaf. For our task of next activity prediction, the output at each leaf is determined through majority voting: the output label y that occurs most frequently in the training samples associated with the leaf is selected as the predicted label.

3.4.4 Cost Complexity Pruning

Depending on the selected stopping criterion, the final DT may include subtrees with numerous nodes that contribute little to the tree's predictive value. To address this, pruning is a common technique used to simplify decision trees by removing unnecessary branches, ultimately improving the tree's performance and generalizability. Additionally, the reduced amount of nodes will make the DT more easily interpretable for human stakeholders.

One effective approach to pruning is **cost-complexity pruning**, which balances the trade-off between a tree's complexity and its predictive accuracy. [**ccp**]

3.5 Fairness

As machine learning systems are increasingly deployed in high-stakes applications, both neural networks and decision trees must be scrutinized for potential biases in their prediction. This concern underscores the importance of having quantitative measures for assessing and comparing models to ensure that they meet fairness criteria alongside performance metrics.

Quantitative group fairness metrics provide a systematic way to evaluate whether a machine learning model behaves equitably across different groups within the population. In this thesis, we focus on one of the most prevalent fairness metrics, **demographic parity**. [**dem_parity**]

4 Methodology

This chapter will outline the complete pipeline of our methodology. It describes the process of data gathering and preprocessing, with a focus on handling bias causing attributes. The chapter then explains how the initial ML model is trained, followed by the application of knowledge distillation to create an interpretable representation of the model, which can then be modified to address unwanted biases. The methodology further includes how the predictions of the modified representation can be used to finetune the original model in order to improve fairness, while maintaining accuracy. The implementation of our methodology is entirely written in python and can be found in [**implementation**].

4.1 Data Gathering

Public real life event logs containing sensitive attributes are hard to come by. Additionally, in order to properly highlight the full capacity of our method, we require the data to have sensitive attributes that cause both positive and negative bias respectively. Although [**simulated_logs**] recently published simulated event logs, seeking to address the scarcity of fairness-aware datasets in PBPM, those event logs didn't show the structure we were looking for in this thesis. Therefore, we will create the necessary data ourselves, either by simulating a process or by enriching a real life event log with sensitive attributes.

4.1.1 Event Log Simulation

Our simulated event logs are generated by following rule-based transitions within a constructed process model. Each case begins at a start event and progresses by iteratively selecting the next activity based on specified transition probabilities, continuing until the end event is reached. Additionally, each case is assigned randomly generated case attributes, which influence certain transition probabilities. The sequences of events will then be saved in the form of an event log, as defined in section 3.1.2. An example for a process model, suitable for simulation, is shown in figure **??**. Transition probabilities along the sequence flow are either defined generally or determined by the value of a specific case attribute. If no sequence flow is defined, the transition probability between activities is set to zero.

4.1.2 Event Log Enrichment

A common critique of using simulated event logs is that their complexity may not fully reflect that of event logs derived from real-life processes. To address this, we augment the publicly available event logs, which lack sensitive attributes, by adding these attributes ourselves. This enrichment process is rule-based: each case is evaluated against specific structural criteria, and attributes are assigned according to distributions that align with the defined rules. Detailed examples of these enrichment rules and their corresponding attribute distributions are provided in section **??**.

4.2 Preprocessing

Since NNs cannot learn from the event log format directly, we preprocess the data to generate samples in the form of (x, y), where $x = (x_1, ..., x_n)$ represents the n input features and $y = (y_1, ..., y_k)$ represents the target outcome out of k activities, as described in section 3.3.4. In order to do this, we extract all possible prefix-outcome pairs from each case in the event log. The prefix consists of the sequence of events up to a certain point in the case, while the outcome corresponds to the next activity after the prefix. Our target y can be directly derived from the next activities by one-hot encoding.

The input x should contain information about both the prefix and the corresponding case attributes. Since neural networks require inputs of a fixed size, we handle the varying lengths of prefixes by applying a sliding window of a static size w. If a prefix is shorter than the window size, we pad the sequence with a special <PAD> activity to fill the remaining positions. Conversely, if a prefix is longer than w, we truncate it to include only the most recent w events.

Case attributes are encoded based on their type. Categorical attributes are one-hot encoded in the same way as activities, ensuring that each category is represented as a unique binary vector. Numerical attributes are normalized using **min-max scaling**, which rescales their values to fall within the range of 0 to 1, ensuring that all attributes contribute equally to the model's training process, preventing numerical attributes with larger magnitudes from dominating those with smaller ones, as well as categorical attributes. Given a numerical value z, the scaled value z' is calculated as

$$z' = \frac{z - \min}{\max - \min} \tag{4.1}$$

where *min* and *max* refer to the minimum and maximum observed values of the attribute in the dataset. Finally, we concatenate the sequence of one-hot encoded activities and the encoded case attributes into the input vector x. An example for the process can be found in figure **??**

After processing all samples, the last step is to split the samples into a **training set** and a **test set**. For this purpose, we used the common train-test ratio of 80-20. The training set is used to improve the NN's accuracy through backpropagation, while the test set is reserved for the final evaluation of the model's performance after training is complete. It is crucial to keep these sets strictly separated to prevent any information from the test set leaking into the training set. Therefore, special care must be taken during the splitting process: [WDW21]

- **Min-Max Scaling:** When performing min-max scaling, the scaling parameters min and max are derived from just the training set. This simulates a real-world scenario in which only the values of old training data is known to us, which ensures that the test set remains independent.

- **Instance-Level Leakage:** We ensure that all events of a single case are contained either in the training set or the test set, but not both. Splitting process instances across sets might lead to information leakage, allowing the model to gain knowledge about test data it would not have access to in a real-world scenario.

- **Temporal Leakage:** When utilizing timestamp date, one could consider using a chronological split to train on earlier cases and test on later ones. This setup mimics real-world predictive scenarios where historical data is used to predict outcomes for future events. In our case, we mostly use simulated event logs, in which there is no chronological difference, or public event logs, in which the timestamps have been anonymized, maintaining only the time difference between events. [Man17] Therefore, we simply used a random split instead.

- **Cluster Leakage:** If the dataset contains groups of highly similar cases, e.g., grouped by customer, department, or product line, assigning entire clusters to either the training or test set prevents correlated information from leaking between the splits. Our datasets however are either simulated or highly anonymized to protect personal information [Man17]. As a result, no distinct clusters are evident, making a random split both appropriate and sufficient.

- **Representative Datasets:** Lastly, it's important to ensure that the test set is representative of the dataset's overall diversity. Specifically, one should avoid

creating splits where rare or common cases are disproportionately represented in one set, as this could skew model evaluation and lead to biased results. Since we are working with rather large datasets, any bias introduced by random splitting is likely to be minimal. Moreover, our primary objective is not to achieve maximum accuracy but to enable a fair and consistent comparison between models, making randomly splitting acceptable for this task.

4.3 Training the Neural Network

After processing the event log into a dataset suitable for machine learning, we proceed to train a feedforward neural network (NN) using the prepared samples. The model is implemented using the **Sequential** class from the Keras library [**keras**], which allows for a straightforward layer-by-layer construction of the network. As mentioned earlier, our primary focus is not on fully optimizing accuracy, so we did not invest extensive time or effort into selecting and fine-tuning the model's hyper-parameters. The following hyperparameters were chosen, because they delivered consistently good results in practice:

The NN incorporates three hidden layers with 256, 128, and 64 neurons The choice of progressively smaller layer sizes allows the network to gradually distill the input features into more abstract representations, facilitating the learning of hierarchical patterns. The amount of perceptrons in the input and output layer is dependent on the task at hand, determined by the dimensions of the processed input and output samples respectively. The hidden layer uses the ReLu activation function, while the ouput layer uses softmax activation, as described in section 3.3.3.

For the training process, we used categorical cross-entropy loss in conjunction with the Adam optimizer, utilizing its default parameters as provided by Keras, with a learning rate of $\alpha = 0.001$. The model was trained for a total of 10 epochs, with a batch size of 32. In our experiments, 10 epochs were sufficient for the model to converge without overfitting. The batch size of 32 was chosen because it offers a practical compromise between computational efficiency and stable gradient updates. This configuration enabled the model to learn effectively and produce satisfactory results within a reasonable training time.

4.4 Knowledge Distillation

After training, the NN is expected to achieve high accuracy but may produce biased predictions, hidden by its black-box nature. To uncover these biases and better

understand the NN's inner workings, we train a transparent model that mimics the NN's behavior.

This approach leverages **knowledge distillation**, a technique originally developed to transfer knowledge from a large, complex model **(teacher)** to a smaller, more efficient model **(student)** [**knowledge_distillation**]. By training the student to replicate the teacher's outputs knowledge distillation enables faster inference and reduced computational demands while preserving predictive performance.

In addition to efficiency, knowledge distillation has been adapted for interpretability. Rather than prioritizing computational simplicity, a transparent student model, such as a decision tree, can be trained to approximate the teacher model's predictions. This allows the decision-making process of the opaque NN to be examined through the interpretable structure of the student, providing insights into its behavior while retaining much of its predictive accuracy.

Although in recent studies more sophisticated white-box models such as [**<empty citation>**], or [**<empty citation>**] have been employed for this purpose, we opted to use regular decision trees (DTs) as defined in section 3.4, since they are both easy to implement and can be interpreted well even by domain experts not too familiar in the field of machine learning.

DTs cannot directly utilize the softened output $\hat{y}$ produced by the softmax activation in the output layer of the neural network, as they require discrete class labels rather than a probability distribution. To address this, we apply the *argmax* function to select the class with the highest probability from the distribution. Formally, given a training sample x, the distilled target labels $\bar{y}$ are derived from the NN's output as follows:

$$\bar{y} = argmax(NN(x)) \tag{4.2}$$

4.5 Training the Decision Tree

Using the input samples x from the training dataset and their corresponding distilled target samples $\bar{y}$, we proceed to train our DT. For this, we employ the **DecisionTreeClassifier** class from the Sklearn library [**sklearn**], which constructs the DT by recursively partitioning the dataset at each node to minimize Gini impurity, as outlined in section 3.4.3.

The training hyperparameters for the DT are selected to prioritize interpretability. The depth of the tree is capped at 10, ensuring the structure remains easy to analyze, while the number of leaf nodes is restricted to a maximum of 50, further managing complexity. To avoid overfitting to small fluctuations in the data, we require each leaf node to represent at least 5 samples, effectively allowing splits only when 10 or more samples are available. Additionally, minimal cost-complexity pruning is applied with $\alpha = 0.001$, eliminating splits that provide negligible reductions in impurity.

4.6 Modification of the Decision Tree

Now that the inner workings of the NN have been represented as a decision tree, we can examine its structure for potential biases. Specifically, we look for inner nodes that use sensitive attributes as the basis for their splits and assess whether such usage is justified in the context of the node. If we determine that one or more nodes unfairly rely on sensitive attributes, we can modify the decision tree structure by deleting those biased nodes, thereby removing their influence. To remove a biased node v while maintaining the defined structure of a binary DT, we have several possible options:

- **Cutting Branches:** Let u_1, u_2 be the children of v and $|S_{u_1}|, |S_{u_2}|$ the number of training samples associated with each child respectively. When removing the node v from the tree, the simplest approach would be to replace v with the child node u_i that has more associated training sampels $|S_{u_i}|$, with $i \in \{1, 2\}$. Basically, we would eliminate the subtree rooted at the child with less samples. This method ensures that the majority of samples are still handled similarly, while removing the biased split from the DT. However, the subtree and its splits may lose their coherence without the biased split, potentially leading to a significant drop in predictive performance after the modification.

- **Retraining Subtrees:** When removing v, an alternative approach would be to delete the entire subtree rooted at v and retrain it using the samples associated with v. To prevent the retrained subtree from making the same biased split, any features related to the sensitive attributes used by v must be excluded from the entire subtree. Although this results in more coherent splits, there is a possibility that the sensitive attribute could have been used by a descendant of v in a beneficial way, which we might not want to eliminate. Consequently, such positive splits would also be prevented.

- **Manual Modification:** Finally, a domain expert could modify the decision tree, its node structure, and the associated features and thresholds based on their

understanding of how the splits should be structured. However, even with domain knowledge, manually adjusting the decision tree requires significant effort and is challenging to do without substantially compromising prediction accuracy.

4.7 Finetuning of the Model

By modifying the DT, our goal was to obtain a predictor that makes fairer decisions, while maintaining most of the NN's predictive power. While we could just use the DT directly for the task of next activity prediction, deep learning methods generally outperform traditional machine learning approaches, when it comes to solving complex tasks [Kra+21]. In order to combine the more precise predictions of the NN with the fairer decisions of the DT, we attempt to fine-tune our NN in accordance to the predictions of the DT.

Fine-tuning is a transfer learning technique that involves adapting a pre-trained model to a new task or dataset. Rather than training a neural network from scratch, fine-tuning starts with a model that has already been trained on a related task, leveraging its pre-learned features. This reduces training time, requires fewer data, and often improves performance by building on the general representations learned during pre-training. [**fine_tuning**]

In our case, we fine-tune the pre-trained NN from the previous section 4.3 using a training set modified to enhance fairness. This fine-tuning dataset retains the same input samples x from the original training set but replaces the original targets y with modified distilled targets $\tilde{y}$, derived from the DT's predictions. However, we can't use the immediate output of the DT, since backpropagation requires a distribution as its target instead of the discrete label provided by the DT. To address this, we apply one-hot encoding, transforming the DT's discrete labels into distribution vectors, similar to the original targets y. The modified distilled target $\tilde{y}$ for a training sample x is computed from the DT's prediction as follows:

$$\tilde{y} = \textit{one-hot}(DT(x)) \tag{4.3}$$

When selecting input samples from the relabeled training set during fine-tuning, we explored several strategies:

- **Selective Sampling**: Only samples where the original target y differed from the modified distilled target $\tilde{y}$ were included. This approach aimed to accelerate the training process by focusing exclusively on the changed targets.

- **Complete Sampling**: All samples were included, even when y and $\tilde{y}$ were identical. This ensured the model retained its pre-learned features, although it required a longer time for the model to adapt to the intended changes.

- **Weighted Sampling**: All samples were presented, but higher weights were assigned to those with changed targets. Samples with heigher weights have an increased impact on the loss function and, consequently, on the learning process. This offered a compromise between the other two approaches.

Since all of these methods yielded identical or comparable results, we opted for selective sampling, focusing only on samples with changed targets, as it offered faster training.

The training process during fine-tuning was similar to the initial training procedure described in section 4.3, with minor adjustments. We did not conduct extensive tests regarding the fine-tuning hyperparameters, since the ideal settings are heavily dependent on both the dataset and the architecture of the black-box predictor. However, these selected hyperparameters yielded satisfactory results in our experiments: We reduced the Adam optimizer's learning rate α from 0.001 to 0.00001 to balance fairness improvements with maintaining the pre-trained weights. Additionally, the number of epochs was reduced from 10 to 5, as the NN quickly achieved perfect accuracy on the modified distilled targets.

5 Evaluation

6 Conclusion

This chapter provides an overview of the key implications derived from our results, discusses the limitations of our approach, and outlines potential directions for future research.

6.1 Implications for theory and practise

One important theoretical implication of our work is the recognition that not all forms of bias in machine learning models should be considered inherently negative. While bias is often viewed as a flaw that must be eliminated, certain biases can reflect domain-specific requirements that contribute to more accurate and context-sensitive predictions. Our experiments have demonstrated that it is possible to remove unwanted biases while preserving beneficial ones, highlighting the need for a more nuanced understanding of bias in machine learning.

From a practical perspective, ensuring fairness in machine learning models is increasingly important, especially in light of legal and regulatory frameworks such as the European Union's General Data Protection Regulation [**eugdpr**] and the Artificial Intelligence Act [**aia**]. These regulations emphasize the need for machine learning systems to be transparent, unbiased, and equitable in their decision-making processes. Our method offers a practical way to incorporate fairness into the model development pipeline, allowing for both transparent evaluation and targeted adjustments to meet fairness standards, without significantly compromising accuracy.

Even in scenarios where fairness-driven changes to the model are not strictly necessary, the insights gained from analyzing and addressing bias can play a critical role in building trust among stakeholders. Understanding how a black-box model behaves and ensuring it aligns with ethical and legal standards can foster confidence in its use, particularly in socially sensitive applications such as hiring, healthcare, and law enforcement. Thus, our fairness-related insights would benefit not only those directly affected by the model's decisions but also the broader ecosystem of users, developers, and regulators.

6.2 Limitations and Challenges

Although our approach holds potential we also noticed some shortcomings while working on this thesis:

- **Decision Tree Splits:** The success of our methods strongly depends on the splits in the decision tree. In order to be able to remove negative bias stemming from a sensitive attribute, without removing the positive bias it has, it is essential for these biases to manifest in different splits. When both kinds of biases are fused into a single node of the tree, they are impossible to separate using our approach. It would be feasible to avoid this by using a modified training algorithm, that penalizes splits based on sensitive attributes, similar to how it was done in [KCP10]. This however might lead to a lower accuracy and transparency, since the decision tree no longer accurately portrays the inner workings of its teacher model, the neural network.

- **Decision Tree Accuracy:**

- **Decision Tree Complexity:** Another problem with higher complexity of tasks, is that the complexity of the decision tree rises as well. When the amount of activities, attributes and their dependencies rises, more splits and therefore nodes are needed in the decision tree to accurately model them, which makes it difficult for domain experts to interpret the decision tree. When limiting the amount of nodes and the depth of the decision tree however, the decision tree remains interpretable, yet may lack in accuracy since it can no longer protray the inner workings of the neural network. This is likely to be a definite bottleneck of our approach, since for highly complex problem there needs to be a tradoff between accuracy and interpretability.

- **Manual Labor:** Finally, compared to other methods from related work such as [QA19], [LP24] or [PDV24], that seek to achieve fairness automatically, based on fixed sets of rules and formulas, our approach requires comparatively more user input from domain experts. Even though it is possible to utilize our method to remove all bias stemming from a sensitive attribute altogether, by removing all nodes that use the sensitive attribute as its split feature, attempting to preserve the positive biases requires a domain expert scrutinizing and carfully modifying the decision tree. This means that in practise, our approach is likely to be more labor-intensive and consequently more cost-intensive than similar methods.

6.3 Future Work

Besides tackling the topics we listed in the previous section 6.2, in order to overcome some of the faced challenges, there are some other topics one could look at for future work.

In this thesis, we focused exclusively on case attributes.

Bibliography

[CH24] Simon Caton and Christian Haas. „Fairness in machine learning: A survey“. In: *ACM Computing Surveys* 56.7 (2024), pp. 1–38 (cit. on p. 5).

[Cha23] Xinyu Chang. „Gender Bias in Hiring: An Analysis of the Impact of Amazon's Recruiting Algorithm“. In: *Advances in Economics, Management and Political Sciences* 23 (2023), pp. 134–140 (cit. on p. 1).

[DAFST22] Maria De-Arteaga, Stefan Feuerriegel, and Maytal Saar-Tsechansky. „Algorithmic fairness in business analytics: Directions for research and practice“. In: *Production and Operations Management* 31.10 (2022), pp. 3749–3770 (cit. on p. 5).

[Goo+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. „Generative adversarial nets“. In: *Advances in neural information processing systems* 27 (2014) (cit. on p. 5).

[KCP10] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. „Discrimination Aware Decision Tree Learning“. In: *2010 IEEE International Conference on Data Mining*. 2010, pp. 869–874 (cit. on pp. 5, 28).

[Kra+21] Wolfgang Kratsch, Jonas Manderscheid, Maximilian Röglinger, and Johannes Seyfried. „Machine learning in business process monitoring: a comparison of deep learning and classical approaches used for outcome prediction“. In: *Business & Information Systems Engineering* 63 (2021), pp. 261–276 (cit. on p. 23).

[LP24] Massimiliano de Leoni and Alessandro Padella. „Achieving Fairness in Predictive Process Analytics via Adversarial Learning (Extended Version)“. In: *arXiv preprint arXiv:2410.02618* (2024) (cit. on pp. 5, 9, 28).

[Man17] Felix Mannhardt. *Hospital Billing - Event Log*. `https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741`. Version 1. dataset. 2017 (cit. on p. 19).

[PDV24] Jari Peeperkorn and Simon De Vos. „Achieving Group Fairness through Independence in Predictive Process Monitoring“. In: *arXiv preprint arXiv:2412.04914* (2024) (cit. on pp. 6, 28).

[PZ19] Victor M Panaretos and Yoav Zemel. „Statistical aspects of Wasserstein distances“. In: *Annual review of statistics and its application* 6.1 (2019), pp. 405–431 (cit. on p. 6).

[QA19] Mahnaz Sadat Qafari and Wil Van der Aalst. „Fairness-aware process mining“. In: *On the Move to Meaningful Internet Systems: OTM 2019 Conferences: Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21–25, 2019, Proceedings*. Springer. 2019, pp. 182–192 (cit. on pp. 5, 13, 28).

[RMVL21] Efrén Rama-Maneiro, Juan C Vidal, and Manuel Lama. „Deep learning for predictive business process monitoring: Review and benchmark“. In: *IEEE Transactions on Services Computing* 16.1 (2021), pp. 739–756 (cit. on p. 9).

[Rud19] Cynthia Rudin. „Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead“. In: *Nature machine intelligence* 1.5 (2019), pp. 206–215 (cit. on p. 9).

[WDW21] Hans Weytjens and Jochen De Weerdt. „Creating unbiased public benchmark datasets with data leakage prevention for predictive process monitoring“. In: *International Conference on Business Process Management*. Springer. 2021, pp. 18–29 (cit. on p. 19).

[Whi04] Stephen A White. „Introduction to BPMN“. In: *Ibm Cooperation* 2.0 (2004), p. 0 (cit. on p. 7).

List of Figures

List of Tables

Declaration

You can put your declaration here, to declare that you have completed your work solely and only with the help of the references you mentioned.

Bayreuth, August 26, 2015

Vorname Nachname